Universidad Carlos III de Madrid

Escuela Politécnica Superior

# Analysis, design and implementation of a storage system in a server with a light interactive interface

Grado en Ingeniería Informática

Trabajo Fin de Grado

Author: Jorge Díaz Gómez

Advisor: Alejandro Calderón Mateos

# INDEX

# TABLE INDEX

# ILUSTRATIONS INDEX

# ABSTRACT

Nowadays all we can see that all the information is growing at an exponential scale because everyday we generate huge quantity of data, from huge companies until normal users of the Internet.

Because of this, the way we interact with that data is getting obsolete and we need to rethink how we can make things easier and accessible.

In that way, though this document you would learn about the generation of a storage system that deals with these problems described.

# 1. INTUDACTION

## 1.1. Motivation

Nowadays we can see that every day data is getting bigger, in an exponential scale. This is happening now because we are used to manage an enormous quantity of data. We can see that web servers need to have huge file systems to be able to store all the content that any user would like to access.

For this reason, many companies and developers are investing money to include these important requirements while bringing new software into the market.

With this intention, this project is a way that I found to make an easy and powerful tool able to collect enormous amount of data from a server used as a storage system while having a craft interface that any user request

The solution that I planned deals with the issue of already fixed interfaces that everybody deals with but none of them have ever questioned. Let make an example to make this clear:

Many people in recent years have their own home webservers, NAS for instance. This tool always comes with a fix and designed interface which is the same in every single NAS. Maybe you can configure some view options, but it doesn't differ a lot from any other NAS.

Users have many different points of view about the real world and this can be a problem in terms of usability. Isn't is easier to let the user decide how they want to see the world without modifying the internal logic of the server?

Another thing that this project deals with is the technology harnessing.

Here we are using an ASUS laptop with an Intel Core I7 to simulate the client programs which connects either to a RaspberyPi-kernel or an external host, the server component.

What we try to do here is separate both worlds (storage and data representation) so we can harness the technology just to what is made for.

To continue with the same example described above, why do we give to the NAS tasks of data representation when our computers can do it better and faster because of they are provided with a better graphic card?

## 1.2. Objectives

In this section I describe which are the three objectives to achieve in this academic project:

- Build a remote file system capable of store as many files and documents as it's storage device space.
- Build a user's interface which can interact with the server.
- Divide functions of tasks to get a better performance in accordance with the technologies being used.

## 1.3. Rest of the document

This section includes a summary of the important parts that this document contains

The next section of the document speaks about an analysis of the state of the art. Here I mention different technologies which already exist on the market related to the project and I´d discuss which is the accurate one for the propose of this thesis. This is done by giving a comparative between some technologies followed by a frame which shows why the ones chosen are more accurate to solve the objectives described above.

The following part contains the analysis phase. Here I go through some similar projects done over the years which has an approximate solution to mine, but don´t completely has the best answer. You can also find here the requirement´s analysis which is one of the most important phases when starting a new project. Here I also include a comparative table which shows whether this solution and why is close or far from the one I proposed. Another thing that you can find here is the user's cases, that is, how the program interacts with the different combinations of inputs.

The following section is the designing phase. Looking at the previous phase here I match everything describe and put it together in components.

The two following sections include the implementation and implantation. while on the first one I speak about the most difficult parts I found while coding the system, on the other I comment how to set-up the project correctly.

The next part is the evaluation one. Here I describe a benchmark and compare my project with Google Drive to visualize the difference between them and take conclusions.

The following part includes the planification, budget and social impact section. On the first part I explain how I planned myself to make this project, this will be reflected on a Gantt graph. On the second section I include the costs derivates from the activity of the project, to finish the las section which includes an analysis of the social impact of the project. And the third part I comment some important social impact that the project may have.

The next part contains some of the important regulations that this project contains where you can find the main standard regulations followed while doing the project. Here you can also find the intellectual properties of the idea.

And finally, I complete the document with the conclusions. I´d mentioned the product that was developed, the process followed and my personal conclusions. I would also include which future assessments could be done in the future to extend the actual function of the system.

# 2. STATE OF THE ART

## 2.1. Introduction

The idea of this project is to build a Remote File System (RMS) which can store different types of elements such as videos, pictures and simple data files able to be passed thought a connexion with a client program, including a simple user interface (UI) to interact with the server.

Many technologies have been studied to make the REST-like server. The main idea found is that many technologies which are used now are either too complex to use or have many programs to be downloaded, or even you need to sign in to any web page to have access to their services.

Because of this I was looking for many already done projects which contains approximates solutions to solve the idea, which I describe on the following sections.

## 2.2. Related Technologies

Now I show some examples of the already technologies whcih are similar to the one proposed.

### *2.2.1. Synology DSM-SO*

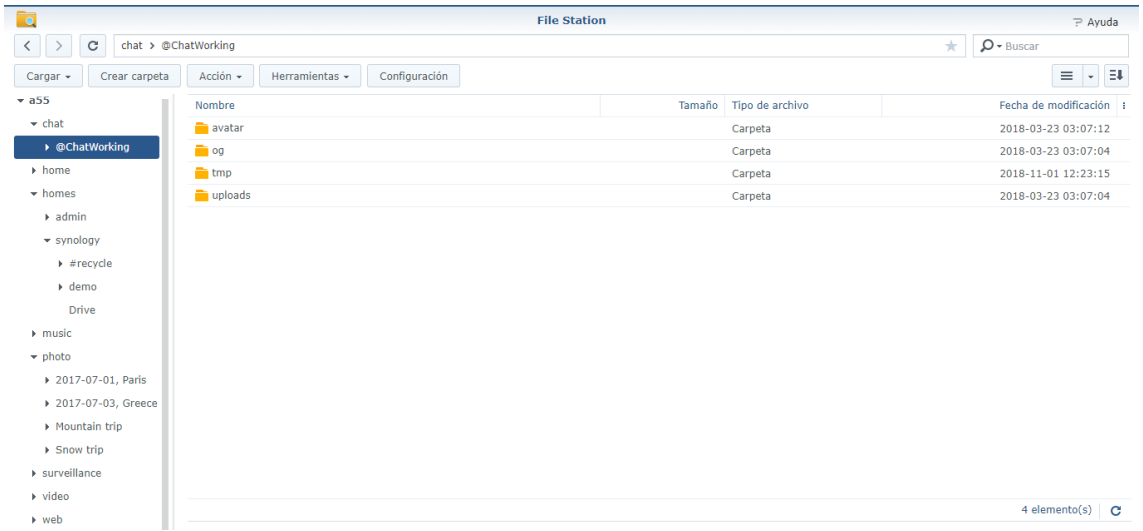


*Ilustración 1:DSM-SO [2]*

Synology is a company created in the year 2000 that offer solutions for data storage in the cloud, IP surveillance and network administration in the cloud. But the main service that this company offers is their Network Attached Storage (NAS).

They´ve been working thought the years trying to create a good storage place for data sharing, synchronization of products and back-ups, in your own home.

The picture above (Ilustracion 1) shows the view of their file system´s UI which was done to control one of their NAS device. I took a trial demo to look around.

I must admit that it looks beautiful and attractive how you move though the UI which is user friendly, but all the graphics are generated on the server component as a process which is displayed there.

I generate the graphics using the capability of my computer, which is better at graphics representation, than using the storage system and 'force' it to do it. This way of thinking reduces the space viability, powering the storage system itself.

### *2.2.2. Google Cloud*


*Ilustración 2: Google Cloud [3]*

Speaking about data storage I can't forget of the major company in the world, which is Google.

Google Cloud is a platform used to stored code and test it in the cloud. It is use for developers to test their codes while is in production. You can set environments to test your code in the cloud.

But besides the brand, I can also criticize things about them.

One of them is its complexity. Google always wants to do everything as big as possible, so they invest too much money to develop it, and for this academic project that is not a valid solution.

Another one is the interaction part, once again a web page is used to navigate thought the services provided. And, as the previous one, we want to generate the graphics in our client program, not in the server.

Finally, you can also see that there are many tools, like database functionalities, proxy setups or even VPN servers which, as mentioned before, are unnecessary for this propourse.

### *2.2.3. Other Google Cloud-like programs*

As well as Google Data studio you also find similar popular tools which their main objective is to store things on it or even make any analytical process with them.

These programs have the same problem as mentioned before, either they use complex functionality, or they add extra functionality which is unnecessary for the objective of doing it “light” apart that they use the server to generate the graphics, which is something that we want to create.

In this group you can find programs like “Dropbox or Google Drive” among others

## 2.3. Comparative table with the project

| Studied Technologies | Computer resources’ explotations | Ease of use | Separation of functions |
|---|---|---|---|
| Synology DSM-SO | ✗ | ✓ | ✗ |
| Google Cloud | ✗ | ✗ | ✗ |
| Dropbox | ✗ | ✓ | ✗ |
| Google Drive | ✗ | ✓ | ✗ |
| My idea | ✓ | ✓ | ✓ |

*Tabla 1-Comparative of studied systems*

# 3. ANALISYS, DESIGN, IMPLEMENTATION AND IMPLANTATION

## 3.1. Introduction

In this section I show how I developed the idea of making this project, including the methodology, the life's cycle and other interesting pieces in relation with the designing process and building process.

The methodology used to make the project is the one that I was though in Software Development lessons.

First is important to complete a clear description of the project´s requirements related to the postulated problem. This is a basic step in any new project because with the requirements you can describe how do you want you program to behave. Here I also study the single technologies that will take part of the project and I analyse the user´s cases which are the steps that users may find while interacting with the platform. This is described in the analysis.

The next step is the designing phase. As on the previous part we analysed all the technologies, now we put them together and create the components that will form the system. Then we create the matrix which will match the requirements to see if we forget any of them.

The final step includes the implementation and deployment parts. On the first one I speak about the difficulties that I found while building the project and on the other part you can find how to run the project.

## 3.2. Analysis

This section contains a study of the already existent technologies which can be part of this project, the requirements and the user's cases

### *3.2.1. Studied technologies*

In this section I would describe the necessary requirements to build the architecture for this project as well as mentioning some of the interesting technologies which already exists and can solve the problem described according to the proposed objectives commented in section 2

This project is dived in two parts. The first part is the server side while the other part is the client and visualization part. Then, I am commenting which technologies I view but for a reason I did not include in my project.

#### 3.2.1.1. Server-side technologies

The technologies listed below are the ones that I found interesting to solve the problem but for any reason I didn't include.

## Case 1: Node-FSAPI

*Ilustración 3: Node-FSAPI [4]*

This API contains a REST server which interacts with a Remote File System. Like all servers of this kind it contains the fundamental operations (GET, POST, PUT, DELETE)

Besides, we can also point out that here you have three security levels. Key level, which is introduced directly in the URL, IP restrictions and HTTPS support which will provide us the use of security certificates.

It also allows the possibility of using different types of parameters such as the listening port, the base directory or the authorization's files control.

This sounds great but in order of building an easy saerver, this is not the best option knowing how much you need to simple obtain a file.

## Case 2: Apache Hadoop

*Ilustración 4: Apache Hadoop [5]*

Apache Hadoop provides an API REST which supports an interface for a Remote File System. As happens with the previous case, you have the basic operations included in any REST-full service: GET, POST, PUT DELETE.

This tool is also used for massive data analysis but in our own case is very complex to be used. For this academic project, we have many functions which are not need so we just left it behind.

#### 3.2.1.2. Client/Visualisation side technology

As I was searching I found an interesting bunch of data representation programs which can be useful to make the visualization and client part.

At the descriptions bellow I show why these tools are not being used in my project

## Case 3: Apache-Zeppelin


*Ilustración 5: Apache Zeppelin [6]*

Apache Zeppelin offers an access notebook via Internet able to make notebooks with interactives graphics. This visualization system helps us to place code to

generate graphics which is helpful if you want to adjust the style of the notebook your own way.

Very useful if you are looking for representation of graphs and data visualization tools but like in our case we are not representing any graphic, just clickable tables there is no need to use Apache Zeppelin, even though this tool could be used as well.

## Case 4: Google Data Studio


*Ilustración 6:Google Data Studio [7]*

Google Data Studio belongs to the google's products for Enterprise use. This group of Google applications contains elements for data analysis (Google Analytics) and Data Visualization. You have access to many data samplers and is comfortable to use because it gives you a sophisticated and user-friendly UI. Although sometimes you can have struggles while using it because it looks very difficult to navigate thought all the options it has.

The main reason why I don't use this platform is because, as we've seen before, is too complex to be used for the proposes we want to archive. This platform is mainly done for big data analytical and AI processes

## Case 5: Microsoft PowerBi

*Ilustración 7 Microsoft PowerBi [8]*

Microsoft presents a data visualization tool, like the one described before, but in shape of a program. You must connect the program to a data sampler and then you can just create many interactives graphs.

This option is viable if our intention is to analyse data, but we are just reading strings and showing tables, so we can interact with the server. This is not our case.

## Extra case: Helio host web server


*Ilustración 8:Helio host [9]*

As I was having problems to run the project in the university in the RaspberianOS, because the Wi-Fi couldn't provide an IP address to the virtual machine, I started to find some free servers to run the project there.

Searching through all the options I found this one which allows Python. This host is free to use for unlimited time and it has many tools such as database maintaining programs, mail host, or even analysis about your data, as well as some obligatory security stuffs to make your program work safely.

### *3.2.2. Requirements*

This section describes the functional and non-functional requirements.

Here we comment two types of requirements, functional and non-functional requirements.

The functional requirements describe at section 3.2.1.1, show the obligatory items that the project needs to work as it must do.

Each functional requirement is generated by its short name RF-X.X, followed with a small description.

The first number represents the type of component it refers to (0 for the server and 1 for the client/visualization) whereas the other number is just the numeration.

In other hand the Non-functional requirements are generated with its short name RNF-X, followed by a small description.

#### 3.2.2.1. Functional requirements

The following tables describe the necessary requirements for the system to work. Here I also include the information related to the libraries that the system must have to work.

| RF-0.1 | |
|---|---|
| **Name:** | There must be a server program |
| **Description:** | The system must have a program which is used as a server. |

*Tabla 2: RF-0.1-Sever program*

| RF-0.2 | |
|---|---|
| **Name:** | Server must be a storage system |
| **Description:** | The server must provide a store functionality cable of contain many different type of elements |

*Tabla 3:RF-0.2-Storage system*

| RF-0.3 | |
|---|---|
| **Name:** | Server must have the flask library imported |
| **Description:** | The Flask library gives us the necessary functions to create the server logic. |

*Tabla 4:RF-0.3-Flask library imported*

| RF-0.4 | |
|---|---|
| **Name:** | Server must have the flask-based functions imported |
| **Description:** | The Flask library gives us the necessary functions to create the server logic. |

*Tabla 5:RF-0.4-flask-based imported*

| RF-0.5 | |
|---|---|
| **Name:** | Server must have the OS library imported |
| **Description:** | The OS library allows us to execute some native Linux instructions. (i.e List of files which a directory has) |

*Tabla 6:RF- 0.5-OS library imported*

| RF-0.6 | |
|---|---|
| **Name:** | Server must have the CSV library imported |
| **Description:** | This library allows us to manipulate CSV-like files |

*Tabla 7:RF-0.6-CSV library Imported*

| RF-0.7 | |
|---|---|
| **Name:** | Server must have the socket library imported |
| **Description:** | This library allows us to work with sockets functions necessary to obtain the IP address where the machine is running. |

*Tabla 8:RF-0.7-Socket library Imported*

| RF-0.8 | |
|---|---|
| **Name:** | Server must have the stat library imported |
| **Description:** | This library allows us to work with the statistics of the files selected |

*Tabla 9:RF-0.8-Stat library Imported*

| RF-0.9 | |
|---|---|
| **Name:** | Server must have the datetime library imported |
| **Description:** | This library allows us to work with dates. |

*Tabla 10: RF-0.9-Datetime library imported*

| RF-1.1 | |
|---|---|
| **Name:** | There must be a client program |
| **Description:** | There must have a program which is used as a client. |

*Tabla 11:RF-1.1-Client program*

| RF-1.2 | |
|---|---|
| **Name:** | Client program must have internet connection |
| **Description:** | The program must run in a host which provides Internet connection |

*Tabla 12: RF-1.2-Internet connection provided*

| RF-1.3 | |
|---|---|
| **Name:** | Client program must have pandas library imported |
| **Description:** | This library allows us to read files which comes from the server. |

*Tabla 13: RF-1.3-Pandas library*

| RF-1.4 | |
|---|---|
| **Name:** | Client program must have the OS library imported |
| **Description:** | The OS library allows us to execute some native Linux instructions. (i.e List of files which a directory has) |

*Tabla 14: RF-1.4-OS Library*

| RF-1.5 | |
|---|---|
| **Name:** | Client program must have the plotly libraries imported |
| **Description:** | This library allows us to work with the graphics. |

*Tabla 15: RF-1.5-Plotly libraries*

| RF-1.6 | |
|---|---|
| **Name:** | Client program must have the ipywidgets imported |
| **Description:** | This library allows us to work with inputs widgets and all the necessary interactional parts. |

*Tabla 16:RF-1.6-Ipywidget library imported*

#### 3.2.2.2. Non-functional requirements

In this section you can find the non-functional requirements

| RNF-1 | |
|---|---|
| **Name:** | Server program is done in Python |
| **Description:** | The server must be implemented in Python. |

*Tabla 17: NFR-1-Server in Python*

| RNF-2 | |
|---|---|
| **Name:** | Server program is placed in a Linux-based OS |
| **Description:** | The program needs to be hosted in a Linux-based OS |

*Tabla 18: RNF-2-Linux-based OS*

| RNF-3 | |
|---|---|
| **Name:** | Client program is done in Python |
| **Description:** | The client program needs to be done in Python |

*Tabla 19: RNF-3:Client in Python*

| RNF-4 | |
|---|---|
| **Name:** | Client program is hosted in a Windows 10 OS |
| **Description:** | The program needs to be hosted in a Windows 10 OS |

*Tabla 20: RNF-4-Client hosted in Windows*

| RNF-5 | |
|---|---|
| **Name:** | URI's are specified on fixed structure: (http://IP address:port/service/[options]) |
| **Description:** | The URI's which the system is provided are fixed expressions |

*Tabla 21: RNF-5: URI´S on fixed structure*

| RNF-6 | |
|---|---|
| **Name:** | The root direction (/) is a valid URI service |
| **Description:** | This root direction provides us information about the services implemented in the system |

*Tabla 22: RNF-6-(/) as root direction*

| RNF-7 | |
|---|---|
| **Name:** | The direction (/files) is a valid URI |
| **Description:** | This direction provides us information about the services implemented in the system |

*Tabla 23: RNF-7- /files is a valid URI*

| RNF-8 | |
|---|---|
| **Name:** | The URI (/{service}/{type}/{nameOfFile.extention}) is valid |
| **Description:** | Directions may have this structure except RNF-8 and RNF-7 |

*Tabla 24: RNF-8- Full URI's has fixed structure*

### *3.2.3. Final Decision*

As we seen above all, the options described are in general right to be used here. The main is either its complexity is too high, which is something we want to avoid, or the unnecessary heavy programs we need to download.

Here you can find the items selected

#### Flask


*Ilustración 9: Flask [10]*

This library contains the main objects to build an API, because this is mainly a library oriented for this use.

The advantages of using it is that is developed for Python, which for me is very easy and fast to learn, if you already know high-level programming languages, and that this library contains a wide range of already programmed functions which are perfect for the propose of building a file server, like sending files, store files, and look for a file with a giving path.

The difference with the other mentioned before is its simplicity of use. You only need to import the library to the main program.

#### Pandas/Matplotlib


*Ilustración 11: Matplotlib [12]*


*Ilustración 10 Pandas [11]*

These libraries are set in the client side. As well as with the previous one, this library contains a huge variety of functions already programmed functions which are very useful to read data and the creation of tables and graphs. Furthermore,

it allows to create plots to include graphs in one page, very useful for data representation and data reading.

## Anaconda


*Ilustración 13: Anaconda [13]*

Anaconda is one open code distribution which contains two of the main data visualization tools languages yet created, R and Python. This use the previous libraries as well as other specifics for R language.

This distribution makes a virtual wrap which is independent from the Operative System (OS) under it (like the new windows 10 terminal that allows to use a Linux console). This is helpful because it allows to use linux based commands in other OS distributions.

Although, it includes a small shell to download any type of library without necessity to make any change within the host OS

## RaspberianOS

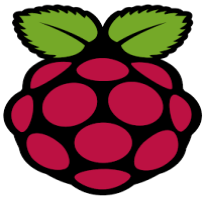

*Ilustración 14 Raspberian [14]*

This OS gives us a GUI which works over a RasperianPI kernel. It follows the logic of the other linux based distributions.

### Jupyer notebook

*Ilustración 15-Jupyter notebook [15]*

Within the Anaconda distribution described in Section 3.2.3.3, Jupyter is an IDE for Python programming.

What makes this notebook interesting is the fact that you can show plots and interaction things on it, which helps the programmer to visualize what is being done. We can say that this one is the easy version of the zeppelin notebook described in section 3.2.2.1

It is also helpful to debug because you can run chunks of code independently.

The following comparative table shows a comparative with the technologies shown against the ones I choose and its positive properties.

| Component | Ease of access | Ease of use | Learning speed | Portability |
|---|---|---|---|---|
| **Case 1**: Node-FSAPI | ✓ | ✓ | ✗ | ✗ |
| **Case 2**: Apache-Hadoop | ✓ | ✓ | ✗ | ✗ |
| **Case 3**: Apache Zeppelin | ✓ | ✓ | ✗ | ✓ |
| **Case 4**: Google Data Studio | ✓ | ✗ | ✗ | ✓ |
| **Case 5**: Microsoft PowerBi | ✗ | ✓ | ✓ | ✗ |
| **Approach**: Flask | ✓ | ✓ | ✓ | ✓ |
| **Approach**: Panda/Matplotlib | ✓ | ✓ | ✓ | ✓ |
| **Approach**: Anaconda | ✓ | ✓ | ✓ | ✓ |
| **Approach**: Raspberian | ✓ | ✓ | ✓ | ✓ |
| **Approach**: Jupyter | ✓ | ✓ | ✓ | ✓ |

*Tabla 25:Comparative with existent technologies*

## 3.3. User's cases

In this section I would comment the user´s cases. As this application is a user's base system, I would describe the cases using the standard BDD.

BDD or Behavour Driven Development is a type of software development technic which is focus in high level, forgetting what is happening at the back end. This kind of agile development is done as if you were at the point of view of the user, which means that all the tests generated afterwards have the user's point of view every time.

The user´s cases will be done using Gherkin expressions, composed by the following statements:

- **Feature**: Name of the functionality we are testing
- **Scenario**: What thing you are testing
- **Given**: Pre-conditions needed to be complete after
- **And**: Task also included in this step
- **When**: Actions of the real testing feature
- **Then**: How the system remains after executing the actions (here we can include how data is stored in the db).

**Featue**: Connection to root data

**Scenario #1**: Giving a correct root address and port

**Given:** Check the server is running

**When**: Place the correct IP:Port direction

**Then**: A frame showing the available services will be displayed

**Scenario #2**: Giving a correct root address with different port

**Given**: Check the server is running

**When**: Place the correct IP but different port address

**Then**: An error appears: "connection was impossible to be made"

**And**: No frame will be displayed

**Scenario #3**: Giving a correct port with different address

**Given:** Check the server is running

**When**: Place the correct port but different address

**Then**: An error appears: “host doesn’t response”

**And**: No frame will be displayed

**Feature**: Service clicking preparation

**Scenario #1**: Clicking at Files service

**Given**: Check the server is running

**When**: Select service at the display menu

**Then**: See the table of files existing on the server

**Scenario #2**: Select search service

**Given:** Check the service is running

**When:** Select “Search” at the display menu

**Then**: See the “find file” input text is displayed

**And**: See type menu is displayed

**And**: See table of files displayed

**Scenario #3**: Select Download service

**Given**: Check the service is running

**When**: Select “Download” at the display menu

**Then**: See the “File name” input text is displayed

**And**: See Data Type menu is displayed

**Feature**: Navigation though Search service

**Scenario #1**: Place a name of an existing file

**Given**: Check the server is running

**And**: Select “Search” at the display menu

**When**: Place “pandas.png” at the “Find file” input tag

**Then**: See a new table appear with the name of the file found

**Scenario #2**: Place a name of a file that do not exist at the file system

**Given**: Check the server is running

**And**: Select “Search” at the display menu

**When**: Place “a.jpg” at the “Find file” input tag

**Then**: No new table appears

**Scenario #3**: Try find pictures at find menu

**Given**: Check the server is running

**And**: Select “Search” at the display menu

**When**: Select “image” at the “Type” display menu

**Then**: The table of the files is updated just with the item selected

**Scenario #4**: Try find data at find menu

**Given**: Check the server is running

**And**: Select “Search” at the display menu

**When**: Select “Data” at the “Type” display menu

**Then**: The table of the files is updated just with the item selected

**Scenario #5**: Try to find all at find menu

**Given**: Check the server is running

**And**: Select “Search” at the display menu

**When**: Select “All” at the “Type” display menu

**Then**: Table updated with all the items stored at the service

**Feature**: Navigation though Download service

**Scenario #1**: Place a name of an existing image to download

**Given**: Check the server is running

**And**: Select “Download” at the display menu

**When**: Select image in Data type menu

**And**: Place “pandas.png” at the “File name” input tag

**Then**: See a new table appear with the name of the file found

**And**: See the massage that the item selected was found

**And**: The file is open in your computer to see what you downloaded

**And**: The item selected is found in the local file system

**Scenario #2:** Place a name of an existing data file to download

**Given**: Check the server is running

**And**: Select “Download” at the display menu

**When**: Select image in Data type menu

**And**: Place “data.json” at the “File name” input tag

**Then**: See a new table appear with the name of the file found

**And**: See the massage that the item selected was found

**And**: The item selected is found in the local file system

**Scenario #3**: Place a name of a file that do not exist at the file system

**Given**: Check the server is running

**And**: Select “Download” at the display menu

**When**: Place “pandas.png” at the “File name” input tag

**Then**: No new table appear with the name of the file found

**And**: No file is open in your computer to see what you downloaded

**And**: No new file found at the local filesystem

**Feature**: Navigation though Upload service

**Scenario #1:** Place a name of an existent file in local system

**Given**: Check the server is running

**And**: Select “Upload” at the display menu

**When**: Place “irrelevant.txt” at the “File name” input tag

**Then**: A message indicating the file was uploaded

**And**: The uploaded file exists in the server

**Scenario #2:** Place a name of an existent file in local system

**Given**: Check the server is running

**And**: Select “Upload” at the display menu

**When**: Place “irrelevant.txt” at the “File name” input tag

**Then**: No message is displayed

## 3.4. Design

In this section I comment how the project was built in accordance with what was said in section 3.2.1 (Analysis)

As I mentioned before, the project is divided in two sections that work together, the server side and the client side as you can see in the user's cases in the previous section.

The following sub-sections will show how the server and the client were design and its structure

### *3.4.1. Server-side component (COM-0)*

The client is made in a Raspbery-py kernel provided by Raspberian. This environment allows us to have access to python programming thought the kernel which give us the opportunity to import the flask library.

This flask library itself contains all the URI´s which are necessary to build the API. As you can see in the last 3 non-functional requirements described in section 3.2.2.2 , all the URI´s have a logical tag which identifies what do they do. So the only thing that the client needs to do is to 'call' one of them and make the necessary operations.

In the version made with heliohost, this component has an .wgsi file. This file is normally inserted in web server to provide security to the requests made. It says which paths of your program are allowed to be accessible by the application, if any request tries to enter to an illegal path, the response would be negative.

### *3.4.2. Client/visualization component (COM-1)*

This system is the one which shows the UI to interact with the storage system as well as acting as the client connecting to the server.

The notebook has the visualization tools packages imported and the only thing it has to do is to connect to the server, process the data being sent and represent it.

Once you interact with the notebook the process calls the URI's and automatically changes as shown in the users cases explained in section 3.3

**Notice**: The importance of the system built is that you only receive small JSON data when interacting with the UI and the only time you get the actual item is once

selected, so the host, in this case my computer, never has extra items which could occupied memory.

### *3.4.3. Comparative table with the component*

Once we described the two components is time to make a comparative table with the requirements described in section 3.2.2 so we can visualize that we didn't forget to include in the solution any of the requirement described.

| | RF-0.1 | RF-0.2 | RF-0.3 | RF-0.4 | RF-0.5 | RF-0.6 | RF-0.7 | RF-0.8 | RF-0.9 |
|---|---|---|---|---|---|---|---|---|---|
| COM-0 | X | X | X | X | X | X | X | X | X |
| COM-1 | | | | | | | | | |

*Tabla 26:Trazability matrix for RF on server*

| | RF-1.1 | RF-1.2 | RF-1.3 | RF-1.4 | RF-1.5 | RF-1.6 |
|---|---|---|---|---|---|---|
| COM-0 | | | | | | |
| COM-1 | X | X | X | X | X | X |

*Tabla 27: Trazability matriz for RF on client*

| | RNF-1 | RNF-2 | RNF-3 | RNF-4 | RNF-5 | RNF-6 | RNF-7 | RNF-8 |
|---|---|---|---|---|---|---|---|---|
| COM-0 | X | X | | | X | X | X | X |
| COM-1 | | | X | X | X | X | X | X |

*Tabla 28: Trazability matrix for RNF*

As we can see above, all the requirements (functional and non-functional) have been satisfied in each component.

## 3.5. Implementation

In this section I would comment which tasks have been more difficult to be implemented.

While I was starting, I tried to make this project differently. My first idea was to do it via web, instead of using an easier UI.

On the first version of the project I found difficulties while trying to create the web page is a tedious work to make, even if you take some examples from Bootstrap. Therefore, I decided to create another version with another functionality implemented on it.

This final project has two versions, the first one that is connected to an image of a raspberryPi and the second one which is hosted in an external server.

Well I had many difficulties to find a free server which was free, even for students. And once I found it, the one I am using now, I had technical problems with them. Either the server was too slow, or the page interaction freeze every time I click to watch my code. I found the solution by emailing them, so they offered me to change the physical server to other one who has a better performance.

Also, and now speaking to the final version, at first, I found many complications while trying to understand the interactions. It was the first time I did something like this so searching on the Internet helps me a lot to clearly understand how it works.

Another issue that I found complicated was how to make the upload service that allows the client program to send data to the server. This is because I was complicating myself too much, when the functionality much easier than I though.

### 3.6. Deployment

This system can be stored in a linux-based OS or any server which allows python programming.

To display the system in a Linux-based OS is necessary first to have the server connected. Then we just need to go to the file where the server is stored and run it with the python command. This will display a message saying the root direction and port where the server is running (the default one is 5000).

If you are using an external host, you can skip the previous step and carry on.

Now we need to connect the client side to the server. First, we need to open the Jupyter notebook app, wait until the file system is open in our default search engine and choose the client program.

Then we just have to click play and follow the instructions appearing in the notebook.

If there is any doubt of how to display the client side, all the necessary documentation for the implantation will be shown at the end of the document, in the section Appendix.

# 4. EVALUATION

In this section I check the performance of the project by benchmarking my version with one of the most used technology commented in section 2, Google Drive.

To do the comparison I am creating a table which shows some general specifications of Google Drive against my system.

The specifications I would use here to compare both technologies will be the following

## 4.1. Speed of download

Any storage system you find out there must have a good speed to download any file, depending its size. For instance, Google Drive will provide you an auto compression if the size of the file or files you want to download is too high.

This metric will be calculated by executing a script that I personally create to calculate the round-trip length of calling a specified URI. This script calculates 10 round-trips and then it returns the average time.

The use of 10 repetitions is because the standard derivation is low so there is no need to take any higher value.

## 4.2. Speed of uploading

This is the opposite option. Imagine you are trying to upload a file an it takes hours to do so, we need to have a good timing to upload files as well.

This metric is calculated by calling the script described above

## 4.3. Number of unnecessary requests

This is something that we don´t realize until we have it in front of us. Imagine that to go from point A to point B we need to make X number of questions. And X is a big number. That is not efficient. This is what this category tries to satisfy.

To do so I will go to the inspect section of any search engine and then I would choose network to see which elements and what is being passed through the network

### 4.4. Quantity of data downloaded

This category is interesting. Imagine that to download a single file, a photo, for example, you download X more unnecessary files. A good file system is the one which only downloads you one single element without any junk traffic behind it.

To do so I will go to the inspect section of any search engine and then I would choose network to see which elements and what is being passed through the network.

.

### 4.5. Software server architecture

As well as metric statistics another import thing is to know whether your application is accessible or not are the static statistics.

This one may show if the server application is REST-alike. This method allows allow communications to be secure, because of the use of HTTPS protocol, as well as it improves its performance

### 4.6. User interface type

One thing that I personally don't like while using Google Drive is the quantity of data that is downloaded, including the loading page

As well as doing the above with the external host, I tried with the host in the VM and it goes fast, it looks that you are accessing via interface, here are the results from the script executed.

| | **Speed of downloading** | **Speed of uploading** | **Request number** | **Quantity of data** | **Software server architecture** | **UI type** |
|---|---|---|---|---|---|---|
| **Google Drive** | 0.3 sec | 2.3 sec | 338 requests | 450KB | REST | Web page interaction |
| **Project** | 1,20 sec | 2.05 sec | 2 requests | 1.7 KB | REST | Local interactions |

*Tabla 29:Benchmark results*

As you can see in the table above, the project can satisfy the main objective of this document, being able to be light.

Even though the downloading speed is a bit higher, the other things has a relative improvement.

Also, if we look to the quantity of data downloaded while doing a request, it looks a big improvement, considering the idea that our system is supposed to be ‘light’

# 5. PLANIFICATION, BUDGET AND SOCIAL IMPACT

In this section I introduce the planification part of the project and the budget of all this associated planification. I would also comment here all technologies used in here and the derivate costs such as light used, personal costs, etc.

## 5.1. Planification

The phases of building the project can be divided in two parts: First the construction of the server and then the construction of the client side. Here I can also add the planification of making this document which was done while all building part was running and done.

Another important thing to comment here is how I build each part. As I comment previously, I followed the normal flow that any software development does, attending to its phases: Analysis, Design, Construction and Evaluation

Bellow I showed a table including time of doing the tasks with its starting and end date:

| Phase | Start date | Complete days | End Date |
|---|---|---|---|
| **Phase1: Analysis** | September-11th | 5 | September-15th |
| **Phase1: Design** | September-16th | 5 | September-20nd |
| **Phase1: Construction** | September-21rd | 10 | Sepember-30th |
| **Phase1: Evaluation** | October-1st | 8 | October-8th |
| **Phase2: Analysis** | October-9th | 5 | October-13th |
| **Phase2: Design** | October-14th | 5 | October-18th |
| **Phase2: Construction** | October-19th | 15 | November-2nd |
| **Phase2: Evaluation** | November-3rd | 8 | November-10th |
| **Memory: Analysis** | November-11th | 9 | November-19th |
| **Memory: Design** | November-20th | 13 | December-2nd |
| **Memory: Construction** | December-3rd | 7 | December-9th |
| **Memory: Evaluation** | December-10th | 7 | December-16th |
| **Memory: State of the art** | December-17th | 5 | December-21th |
| | | Total: 102 | |

*Tabla 30:Project planification*

To show a visual planification I also give the Gantt graphic associated to the table:

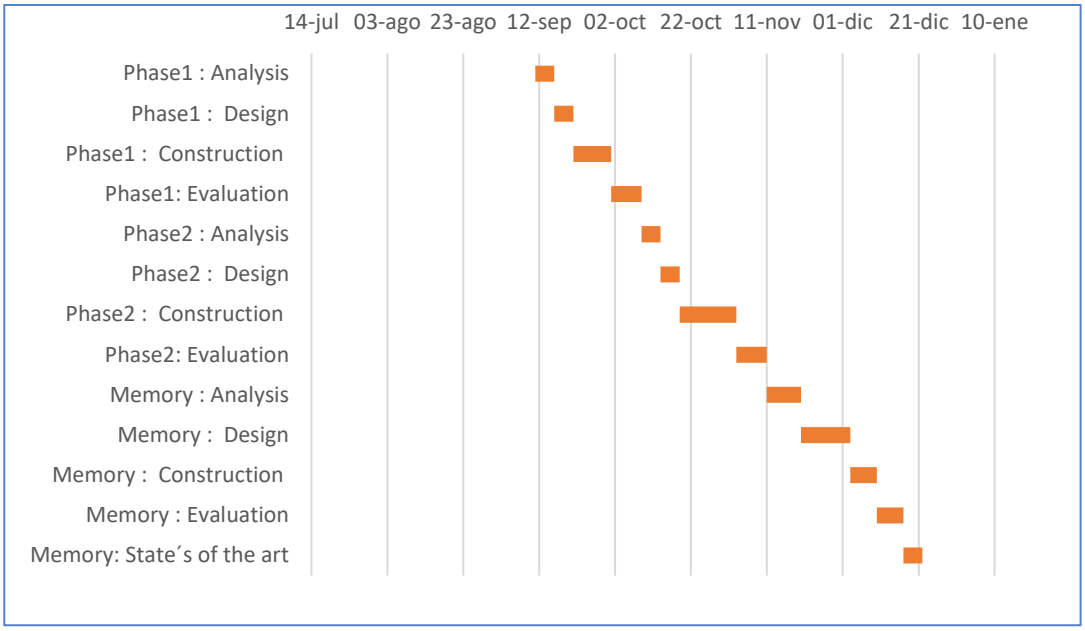


*Tabla 31: Gantt Chart*

The remaining time over December was to make some changes both in the code and memory according to some revisions that weren't planed initially.

## 5.2. Budget

Attending to the tasks shown in the previous section I can estimate the budget of all the days.

The estimation of the budget is done for the use of hardware, software, and working personals who took part during the development of this project.

The length of the project was estimated to be 102 days, but in hours is estimated to be between 204 and 306 hours, an average of 2.5 hours of work a day.

The following table shows the estimated cost of the material:

| Category | Element | Number | Price/Number | Total (€) |
|---|---|---|---|---|
| Hardware | Computer | 1 | 600 | 600 |
| Software | | | 0 | 0 |
| Materials | A4 Paper pack | 1 | 3 | 3 |
| Total | | | | 603 |

*Tabla 32:Costs of materials*

As you can see, all the software shown above is 0 because I either use GNU/linux OS which has a free license or Windows that was included in the total price of the computer.

The cost for the hosting server was also free because heliohost is a free serving hosting.

The next frame shows the extra indirect cost derived from the exercise of this project:

| Element | Price/Month | Total (€) |
|---|---|---|
| Light | 35,48 | 141,92 |
| Commuting | 20 | 80 |
| Total | | 221,92 |

*Tabla 33:Indirect costs*

This lightning cost was estimated by searching the average cost of light of the year

The next table shows the human cost:

| Job | Person/hour(AVG) | Price/hour | Total (€) |
|---|---|---|---|
| **Analyst** | 125 | 49 | 6125 |
| **Designer** | 125 | 50 | 6250 |
| **Programmer** | 125 | 43 | 5375 |
| **Tester** | 200 | 30 | 6000 |
| **Total** | | | 23750 |

*Tabla 34: Human costs*

This table was calculating by searching each profile working on the project as shown in the table Gantt Chart of the previous subsection.

The salary was calculated by searching the average hour payments and adding the total estimated hours that one of them 'took part' in the project

Finally, the last table shows the total cost of every table, giving that this project has an estimated cost of 35.839,92 €

| Type | Total (€) |
|---|---|
| **Materials** | 603 |
| **Indirect costs** | 221,92 |
| **Personal** | 35015 |
| **Total** | 35839,92 |

*Tabla 35.Total Cost*

### 5.3. Analysis of the social impact

In this section I comment which social impact this project has.

First, and most important, the electricity cost. As you can see in Ilustración 17 [1], at the bottom of this page, there is a comparative of the power consumption when it is active (dark blue colour) and in sleep mode (light blue). Our studied case is the comparison between the desktop PC and the laptop/notebook, which is where I built the system.

If our intention is to maintain the server available 24/7, the cost will be reduced because it consumes less electricity while is operative.

Another thing that this system will help is to reduce the human cost. Imagine that at any point the server stops, and we need a technician to fix it. As the code is simple it could be easy if any bug appears to know where the problem is and solve it, which will require less hours trying to find where the bugs are.

The same thing would happen with its maintenance. Simple code makes it easy to check and therefore to learn, if the maintenance of the whole system needs less control you can even do it by yourself and avoid hiring a new person to do the job for it.

To sum up, for a company or even personal use, this kind of functionality is perfect.

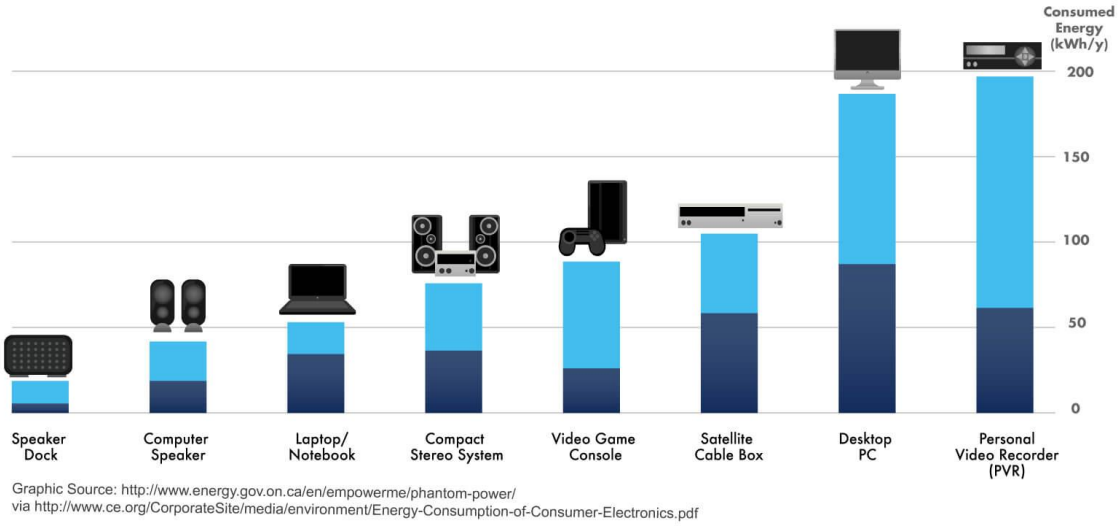


*Ilustración 16:Comparave of power consumption [1]*

# 6. REGULATORY ENVIRONEMENT

In this section I comment the most important regulatory environments that this project has.

## 6.1. Analysis of the applicable law

In this section I mention which are the most important laws that are applicable according to the matter of my project.

### *6.1.1. Risks*

In this section I mentioned the risks that this project may have.

Because we are storing things in a server one of the main problems that we can find here is a blackout that may make the server unreachable, so clients can't access to their data.

Another important thing to attend is the possibility of a destruction or damage of the server because of the effect of a natural disaster like an earthquake or a flooded.

### *6.1.2. Security*

In this section I mention the most important security regulatory specifications made to mitigate the risks described in the previous section.

The natural catastrophes are frequent in our world. So one possible mitigation effect decision is to have two servers in different places so if any catastrophe happens, you can restore the data in it.

Another risk that can be possible to be mitigated is the case of blackout. If you have any security copy of the files anywhere else, you can redirect the traffic while one server is lost.

These security risks correspond to the regulatory standards ISO/IEC.

### *6.1.3. OLDP*

This regulatory requirement is in all the projects which include data storage, like Facebook or Google Drive.

As a storage system, we need to be careful what do we do to the data stored in the server component. That is why with our implementation we solve the problem that anyone external could access to the server and steal personal.

We would also inform all the users that this platform may have in the future (because now is just a prototype) that we won't share any personal data to any third company.

The concepts described here belongs to the new mandatory regulation imposed in Europe in the year 2018.

## 6.2. Intellectual property

Here I describe the intellectual property of the idea.

The idea of this project is subject to the Creative Commons BY-NC-SA which implies the following conditions

1- -You can download and share the project if you mentioned the original author.
2- You can't use the project for commercial use.
3- You can modify, remix and transform the project but the new product obtained must have the same license as the original author.

You can check more information about this license in the following link

# 7. CONCLUSIONS AND FUTURES ASSESMENTS

## 7.1. Conclusions

### *7.1.1. Conclusions of the product*

**Build a remote file system capable of store as many files and documents as it's storage device space.**

To archive this objective, I built a server to be stored in a RaspberryPi. This device is good at storing things so the decision of building it was succeeded.

Also, with the help of Python programming as well as their specified libraries, in this case for creating a REST API, was a master piece to reproduce the storage system as an API.

**Build a user interface which can interact with the server.**

The facility obtained with the use of Jupiter notebook is key to archive this objective because the widgets being used to make the interactions part as well as printing it at the same paper, helped a lot.

As we go through the interface you can select all your data and either uploading or downloading as you want. This is what makes the project unique in my opinion.

**Divide functions of tasks to get a better performance in accordance with the technologies being used.**

This objective was the combination of the above ones which I also satisfied. This objective was done by putting the storage component in a RaspberyPi or the external server, and then the graphics and GUI function attached to my laptop.

This separation of functions was possible to be made thanks to the characteristic that the server has a software REST-like architecture. As nowadays this is getting more important, many languages include this functionality already programmed, you only need to understand the theory.

The separation obtained was crucial to achieve what we described in section 5.3 (Social Impact).

### *7.1.2. Conclusions of the process*

All problems happened during the construction stage of both systems together. This issue made me go slower than I previously thought. The main issue that happened was that at first, I didn't know how to use some of the features that the Flask package contains, or some of the package used to make the representation of the graphics at the computer. But after looking for some information and some examples of how to use them, it became easily for me, so when I did previous extensions of both programs, I found them less tough.

This project was developed just by me. It took four complete months to be done, but I don't think this project would have taken less time if it would have done by more people because in any way both systems are connected. Imagine that someone finishes the server side but other stills doing the client side. If you don't certainly know which values and how the values are returned or sent to the server, you can't continue with the next step.

### *7.1.3. Personal Conclusions*

This project was easier to do thanks to some subjects that I will be revising next:

- **Users Interfaces:** In this subject I learnt how to build efficient UIs so that users could use it in a fast and easy way, and I also learnt many types of UI´s that already exists. During the development of this project, I found it useful while I was trying to thing how the visualization of data could be done.
- **Computer networking:** As this project has connections involved this subject was also useful. Here I learnt the Internet stack and revise deeply though all the important things every part has. While I was trying the IP connections with the app, I thought that It was easy to do and understand just because at this subject I learnt how to do it.
- **Web computer´s technologies:** Here we basically learnt how to build a full front-end/back-end web. One of the concepts that I learnt here was the concept of the API and the REST paradigm. This was useful to build all the server part, which is a RESTful server API.
- **Computer networking:** This subject also influenced me while developing. Here we learnt all the basic syscalls that any OS must do to make it works as

it should. Here he learnt about the file system's handling where in my project I use it to create the necessary files.

- **Artificial Intelligent:** An interesting subject to learn the fundamentals of AI. Here I basically learnt many AI algorithms and I built an auto-pacman player who chooses the best way to avoid the ghost. The interesting part was that this assessment was all in Python, so there was where I learnt the fundamental language that I use in this project.

On the other hand, I learnt how to use other programs where I have never been told about, some examples that I can extract here is the use of the Flask library, all the libraries to represents the data and build the UI, and the amazing world of Jupyter notebook, used for data representations in the big data field.

## 7.2. Future works

Here I am commenting some future assessments that will be possible to include in order of improving the system I created.

- **Security:** As this tool is just a prototype, I only include a wgsi file to control the connections to the external server, but I didn't include any security part, except for the ones which already come integrated in the tools being used. What we can try to improve here is including any kind of identification part to start the system (sign in/login/ etc).
- **Users:** Another improvement that can follow the previous one is the possibility of using users and sessions, so many people can have separate spaces to store their files.
- **Services provided:** A wide range of more services can be implemented to make the system better, for example we can add another module which is able to connect to an external API to obtain information related to the climate, sports, or this short of things.
- **Include more functions to update the storage system:** This is another improvement that can be done. As you can see here, now we can connect and download, and upload files but we can't modify anything stored, like delete a file or even store a heavy file, like a video.
- **Create a mobile interface to connect to the server:** Another extra feature that could be included here is the possibility to access the data via mobile. Nowadays many programs have their equivalence to portable devices.

# Appendix

In this section you could find the full deployment of the code. If you want to see the original code then go to https://github.com/jorgediazgomez96/tfg-project and this may redirect you to the git repository where the code is.

## How to set up the system

The client side of the application was displayed in Windows 10 using Jupyter notebook. The server side of the application was displayed in a Raspberian VM with Python installed. First clone the repository in two OS, one with Linux program (server.py and directorios) and the other (clients) in Windows10 incorporated in Jupyter notebook.

git clone https://github.com/jorgediazgomez96/tfg-project

#### WHAT TO DO IN SERVER SIDE

Go to the following direction: ./tfg-project and start the server( python server.py) On the screen you must see a message like "Server is running at the following direction "http://IPaddress:port"

#### WHAT TO DO IN CLIENT SIDE

Here you need open first your Jupyter notebook. If your system doesn´t have it, check how to install Jupyter notebook on your computer. Another important thing is to make sure your system has the required libraries installed to make the project run without errors

Check which at the top of the page which are the libraries being used and download them with (pip <library-name>.

**Note:** pip is the python package downloader, like npm in node. If you are using Anaconda go to the Anaconda's shell and download it, will be ready on your project.

#### WHAT IF YOU WANT TO TRY ONLINE

Recently I created a domain in heliohost.org. They provide free domains to run small projects like mine.

So, if you want to change from local to online, you just follow the following steps:

1. Open clients/newInteractiveTable.ipynb on your Jupyter notebook.
2. Modify the variable Piserver tohttp://filesystem.heliohost.org/flask
3. Create a new variable, for example serviceServer and type the following address: http://filesystem.heliohost.org/flask/services

4. Put the created variable at the line where you see: pd.read_csv( ) so the new sentence would be pd.read_csv(serviceSerer)
5. Execute normally and if nothing strange happens it would work perfectly.

## How to make an URI test

As described in section 4 (EVALUATION) to try some of the characteristics for the benchmark I created a script.

**INSTRUCTIONS**

1. Clone this project
2. Select the URI you want to test
3. Select the necessary flag

**EXTRA EXPLANATION**

The flag here has three possible values:

1- 'csv' if you are reading pandas csv files alike

2- 'image' if you are using the normal requests python package (I am downloading an image and I want to check the speed of it)

**3-** 'json' if you are reading pandas json files

**OUTPUT**

In every link you are checking first it will display a single time for the URI inserted and then it will calculate the AVERAGE of the times. It is set to 10 the number of times calculated for each URI.

# BIBLIOGRAPHY


[1] www.hidorone.com, «Is your home hauted?,» *Hidroone.*

[2] Erik, «Using virtual synology in a scale out distributed storage architecture,» *Erik Bussink: Technology & Rants,* 2015.

[3] A. Li, «Google Cloud establishes new polices to prevent & delay suspensions, increasing support,» *9to5Google,* 2018.

[4] Fluidbyte, «A RESTful FileSystem API for NodeJS,» *Github,* 2016.

[5] R. B. TI, «Apache Hadoop cumple 10 años,» *ByteTi,* 2016.

[6] D. Ziganto, «How To Locally Install & Configure Apache Spark & Zeppelin,» *Standard Derivations,* 2017.

[7] M. Heymann, «Google Data Studio: ¿tienes problemas conectando con MySQL? ¿Pasa algo con la zona horaria?,» *The ducks in a row,* 2017.

[8] «Microsoft Powr BI,» *Compra Software.*

[9] I. Sammy, «Heliohost,» 2017.

[10] S. Sareen, «Flasks in Python,» *Medium,* 2018.

[11] A. Bronshtein, «A Quick Introduction to the “Pandas” Python Library,» *Throwards Data Science,* 2017.

[12] L. Salcedo, «Visualización de Datos con Python y Matplotlib,» *Mi diario de Python.*

[13] M. I. L. Quinter, «Anaconda – Distribución Python,» *Linux Hispano,* 2016.

[14] L. Adictos, «Raspbian OS añade soporte para GPU en OpenGL,» *Tequila Valley Cancún,* 2016.

[15] H. Flatau, «An Overview of Jupyter Notebooks,» *Software Engineering Daily: The world through the lens of software,* 2018.